華中科技大學

课 程 实 验 报 告

课程名称：C++程序设计

实验名称：面向对象的整型队列编程

院　　系 ：计算机科学与技术

专业班级 ：　CS2006

学　　号 ：　U202015471

姓　　名 ：　杨释钧

指导教师 ：　纪俊文

2021 年 12 月 12 日

目录

1 需求分析

1.1 题目要求

整型队列是一种先进先出的存储结构，对其进行的操作通常包括：向队列尾部添加一个整型元素、从队列首部移除一个整型元素等。整型循环队列类 QUEUE 及其操作函数采用面向对象的 C++语言定义，请将完成上述操作的所有如下函数采用 C++语言编程， 然后写一个 main 函数对队列的所有操作函数进行测试，请不要自己添加定义任何新的函数成员和数据成员。

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
class QUEUE{
      int* const   elems;  //elems 申请内存用于存放队列的元素
   const   int   max;      //elems 申请的最大元素个数为 max
   int     head, tail;     //队列头 head 和尾 tail，队空 head=tail;初始 head=tail=0
public:
   QUEUE(int m);        //初始化队列：最多申请 m 个元素
   QUEUE(const QUEUE& q);              //用 q 深拷贝初始化队列
   QUEUE(QUEUE&& q)noexcept;           //用 q 移动初始化队列
   virtual operator int() const noexcept; //返回队列的实际元素个数
   virtual int size() const noexcept;     //返回队列申请的最大元素个数 max
   virtual QUEUE& operator<<(int e);      //将 e 入队列尾部，并返回当前队列
   virtual QUEUE& operator>>(int& e);     //从队首出元素到 e，并返回当前队列
   virtual QUEUE& operator=(const QUEUE& q);//深拷贝赋值并返回被赋值队列
   virtual QUEUE& operator=(QUEUE&& q)noexcept;//移动赋值并返回被赋值队列
   virtual char * print(char *s) const noexcept;//打印队列至 s 并返回 s
   virtual ~QUEUE();                          //销毁当前队列
}
```

编程时应采用 VS2019 开发，并将其编译模式设置为 X86 模式，其他需要注意的事项说明如下：

（1）用 QUEUE(int m)对队列初始化时， 为其 elems 分配 m 个整型元素内存，并初始化 max 为 m，以及初始化 head=tail=0。

（2）对于 QUEUE(const QUEUE& q)深拷贝构造函数，在用已经存在的对象 q 深拷贝构造新对象时，新对象不能共用已经存在的对象 q 的 elems 内存，新对象的 elems 需要分配和 q 为 elems 分配的同样大小的内存，并且将 q 的 elems 的内容深拷贝至新对象分配的内存；新对象的 max、head、tail 应设置成和 q 的对应值相同。

（3）对于 QUEUE(QUEUE&& q)移动构造函数，在用已经存在的对象 q 移动构造新对象时，新对象接受使用对象 q 为 elems 分配的内存，并且新对象的 max、head、tail 应设置成和对象 q 的对应值相同；然后对象 q 的 elems 设置为空指针以表示内存被移走，同时其 max、head、tail 均应设置为 0。

（4）对于 QUEUE& operator=(const QUEUE& q)深拷贝赋值函数，在用等号右边的对象 q 深拷贝赋值等号左边的对象时，等号左边的对象若为 elems 分配了内存，则应先释放内存以避免内存泄漏。等号左边的对象不能共用对象 q 的 elems 的同一块内存，应为其 elems 分配和 q 为 elems 分配的同样大小的内存，并且将 q 的 elems 存储的内容拷贝至等号左边对象分配的内存；等号左边对象的 max、head、tail 应设置成和 q 的对应值相同。

（5）对于 QUEUE& operator=(QUEUE&& q)noexcept 移动赋值函数，在用已经存在的对象 q 移动赋值给等号左边的对象时，等号左边的对象若为 elems 分配了内存，则应先释放内存以避免内存泄漏。等号左边的对象接受使用 q 为 elems 分配的内存，并且等号左边的对象的 max、head、tail 应设置成和对象 q 的对应值相同；对象 q 的 elems 然后设置为空指针以表示内存被移走，同时其 max、head、tail 均应置为 0。

（6）队列应实现为循环队列，当队尾指针 tail 快要追上队首指针 head 时，即如果满足 (tail+1)%max=head，则表示表示队列已满，故队列最多存放 max-1 个元素；而当 head=tail 时，则表示队列为空。队列空取出元素或队列满放入元素均应抛出异常，并且保持其内部状态不变。

（7）打印队列时从队首打印至队尾，打印的元素之间以逗号分隔。

1.2 需求分析

本实验要求利用面向对象的编程方法实现一个队列类，要求将队列实现为顺序的循环队列，应当实现队列具备的相关的操作。

2 系统设计

2.1 概要设计

面向对象的队列编程的总体思路其实很简单，用一个数组存放队列中的元素，类中包括四个实例数据成员，即整形指针、队头和队尾的位置以及队列容量，主要包括了队列类的创建及相关操作的实现模块，类的声明以及每一种方法的作用如下：

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<string>
#include<stdlib.h>
class QUEUE {
    int* const elems;//elems申请内存用于存放队列的元素
    const int max; //elems申请的最大元素个数为max
    int head, tail;        //队列头head和尾tail，队空head=tail;初始head=tail=0
public:
    QUEUE(int m);        //初始化队列：最多申请m个元素
    QUEUE(const QUEUE& q);          //用q深拷贝初始化队列
    QUEUE(QUEUE&& q)noexcept;       //用q移动初始化队列
    virtual operator int() const noexcept;     //返回队列的实际元素个数
    virtual int size() const noexcept;         //返回队列申请的最大元素个数max
    virtual QUEUE& operator<<(int e);        //将e入队列尾部，并返回当前队列
    virtual QUEUE& operator>>(int& e);       //从队首出元素到e，并返回当前队列
    virtual QUEUE& operator=(const QUEUE& q);//深拷贝赋值并返回被赋值队列
    virtual QUEUE& operator=(QUEUE&& q)noexcept;//移动赋值并返回被赋值队列
    virtual char* print(char* s) const noexcept;//打印队列至s并返回s
    virtual ~QUEUE();                          //销毁当前队列
};
```

循环队列示意图如图2.1所示。

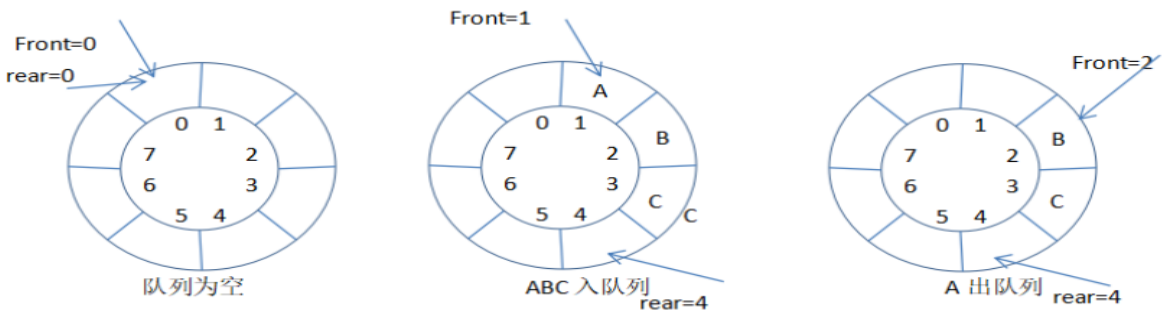

图2.1

2.2 详细设计

队列的数据成员包括指向用于存放队列的元素内存的整形指针 elem，申请的最大元素个数 max，指示队首与队尾的整形变量队列头 head 和尾 tail。函数成员及其说明如下：

(1)初始化队列：最多申请 m 个元素 QUEUE(int m)：先为 elem 申请一块可以存放 m 个整形元素的内存，然后将 max 设置为 m，此时便完成了队列的创建，最后将 tail 与 head 设为 0，表示初始状态队列为空，队列初始化完成。

(2)用 q 深拷贝初始化队列 QUEUE(const QUEUE& q)：深拷贝初始化要求新对象的 elems 需要分配和 q 为 elems 分配的同样大小的内存，并且将 q 的 elems 的内容深拷贝至新对象分配的内存；新对象的 max、head、tail 应设置成和 q 的对应值相同。因此先为 elems 分配一块大小为 q.max 大小的内存，然后将 max，head，tail 分别设为 q.max，q.head，q.tail，最后遍历 q 将 q 中所有元素赋给新对象的对应下标的元素。

(3)用 q 移动初始化队列 QUEUE(QUEUE&& q)noexcept：移动初始化要求新对象接受使用对象 q 为 elems 分配的内存，并且新对象的 max、head、tail 应设置成和对象 q 的对应值相同；然后对象 q 的 elems 设置为空指针以表示内存被移走，同时其 max、head、tail 均应设置为 0。因此直接将 elems 指向 q.elems，然后将 max，head，tail 分别设为 q.max，q.head，q.tail，最后将 q.elems 设为 nullptr，将 q.max，q.head，q.tail 设为 0。其中由于 elems 和 max 都是只读类型的变量，所以修改其值时需要先取地址，然后进行指针强制类型转换，最后再访问指针指向的内存即可。

(4)返回队列的实际元素个数 virtual operator int() const noexcept：由于队列时循环队列，故 tail 可能小于 head，因此队列的元素个数应为(tail+max-head)%max。考虑到 max 可能为 0，故只有再 max 不为 0 的时候返回(tail+max-head)%max，若为 0 则直接返回 0。

(5)返回队列申请的最大元素个数 virtual int size() const noexcept：直接返回 max 即可。

(6)将 e 入队列尾部，并返回当前队列 virtual QUEUE& operator<<(int e)：入队前先要判满，循环队列满的条件为若再入一个元素则 tail 与 head 相等，即(tail+1)%max==head。若队列满则抛出异常，若队列未满则将元素放入 elem 中下标为 tail 的内存，最后改变 tail，根据循环队列的规律，tail 应该改变为(tail+1)%max。

(7)从队首出元素到 e，并返回当前队列 virtual QUEUE& operator>>(int& e)：出队前先要判空，循环队列空的条件为 tail==head。若队列空则抛出异常，若队列不为空则将下标为 head 的值赋给 e，然后改变 head 的值，根据循环队列的规律，head 的值应该改变为(head+1)%max。

(8)深拷贝赋值并返回被赋值队列 virtual QUEUE& operator=(const QUEUE& q)：首先若传入的队列若就是自己，即 elem==q.elem，则没必要进行后面的操作，直接返回*this 即可。若 this 也被初始化过，即 elem 不为空，由于之后要对其分配新的内存使其指向的内存大小与 q.elem 相同，为防止内存泄漏，先释放 elem 的内存将其设为空，后面的操作与深拷贝初始化一致。

(9)移动赋值并返回被赋值队列：QUEUE& QUEUE::operator=(QUEUE&& q) noexcept：首先若传入的队列若就是自己，即 elem==q.elem，则没必要进行后面的操作，直接返回*this 即可。若 this 也被初始化过，即 elem 不为空，由于之后要将其指向 q.elem，为防止内存泄漏，先释放 elem 的内存将其设为空，后面的

操作与深拷贝初始化一致。

(10)打印队列至 s 并返回 s virtual char* print(char* s) const noexcept：基本思路为先将 s 设为空船，然后从 head 开始遍历队列，将每个元素与一个逗号写入临时变量字符数组 a 中，然后将 s 与 a 拼接即可。遍历时将循环变量 i 初始化为 head，由于 tail 和 head 的大小关系不确定，故不可以采用大小比较来确定是否遍历完成，所以设置循环终止条件为 i !=tail，每遍历一个元素 i 更新为(i+1)%max。

(11)销毁当前队列 virtual ~QUEUE()：释放 elem 内存后将数据成员全部设为 0 即可。

3 软件开发与测试

3.1 软件开发

硬件环境：

- 处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
- 机带 RAM：16.0GB(15.4GB 可用)
- 系统类型：64 位操作系统，基于 x64 的处理器

开发环境：MSVC 2019 C++11，编译模式为 X86。 部分代码在 gcc， gdb 环境调试完成。

3.2 软件测试

测试结果如图 3.1 所示。

图 3. 1

测试结果良好。

4 特点与不足

4.1 技术特点

本次实验采用面向对象的编程思想实现了一个先进先出的循环整形队列，整体代码较为简单。

4.2 不足和改进的建议

队列为定长队列，泛用性差，可以改进为动态分配的内存的队列，即在队列满时调用 realloc 函数为队列重新分配一块更大的内存空间同时更新 max 的值，也可以使用链式队列，以提升空间利用率。

此外，由于已经使用了面向对象编程方法，可以考虑增加模板，不仅仅实现整形队列，也可以去支持一些别的类型，适当增加一些功能，进而提升实用性。

5 过程和体会

5.1 遇到的主要问题和解决方法

1. 首先，由于自己在做实验的时候对 C++语言还不是很熟悉，不是很清楚在类的私有成员中含有 const 类型时如何对其初始化，导致出现问题。后来经过查阅资料与询问同学得知需要采用初始化列表的方法对 const 类型进行初始化。

2. 其次，在做实验的过程中，由于我没有在析构的时候检查指针是否为空，所以有可能会造成重复析构的问题。最终在老师的指导下成功的发现并更改了这一问题。

5.2 课程设计的体会

本次实验采用面向对象的编程思想实现了一个循环整形队列，让我对 C++以及面向对象的编程思想有了初步的了解，同时对一些编程规范有了一定的认知。

6 源码和说明

6.1 文件清单及其功能说明

提交文件中，def.h 包含了整形队列类的声明，queue.cpp 文件包含了整形队列类的成员函数的定义，main.cpp 函数包含了测试文件。

6.2 用户使用说明书

使用 vs2019 打开并在本地运行即可。

6.3 源代码

def.h:

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<string>
#include<stdlib.h>
class QUEUE {
    int* const elems;//elems申请内存用于存放队列的元素
    const int max; //elems申请的最大元素个数为max
    int head, tail;        //队列头head和尾tail，队空head=tail;初始head=tail=0
public:
    QUEUE(int m);        //初始化队列：最多申请m个元素
    QUEUE(const QUEUE& q);          //用q深拷贝初始化队列
    QUEUE(QUEUE&& q)noexcept;       //用q移动初始化队列
    virtual operator int() const noexcept;     //返回队列的实际元素个数
    virtual int size() const noexcept;         //返回队列申请的最大元素个数max
    virtual QUEUE& operator<<(int e);        //将e入队列尾部，并返回当前队列
    virtual QUEUE& operator>>(int& e);       //从队首出元素到e，并返回当前队列
    virtual QUEUE& operator=(const QUEUE& q);//深拷贝赋值并返回被赋值队列
    virtual QUEUE& operator=(QUEUE&& q)noexcept;//移动赋值并返回被赋值队列
    virtual char* print(char* s) const noexcept;//打印队列至s并返回s
    virtual ~QUEUE();                          //销毁当前队列
};
```

queue.cpp:

```
#include"def.h"
QUEUE::QUEUE(int m):elems(new int[m]), max(m)
{
    tail = 0;
    head = 0;
}
QUEUE::QUEUE(const QUEUE& q) :elems(new int[q.max]), max(0)
{
    for (int i = 0; i < q.max; i++)
    {
        elems[i] = q.elems[i];
    }
    *(int*)&max = q.max;
    head = q.head;
    tail = q.tail;
}
QUEUE::QUEUE(QUEUE&& q)noexcept:elems(q.elems), max(q.max)
{
    head = q.head;
    tail = q.tail;
    q.tail = 0;
    q.head = 0;
    *(int**)&q.elems = NULL;
    *(int*)&q.max = 0;
}
QUEUE::operator int() const noexcept
{
    if (elems == NULL)
    {
        return 0;
    }
    return (tail + max - head) % max;
}
int QUEUE::size() const noexcept
{
    return max;
}
QUEUE& QUEUE::operator<<(int e)
{
    if ((tail + 1) % max == head)
    {
        throw("QUEUE is full!");
        return *this;
    }
    elems[tail] = e;
```

```
    tail = (tail + 1) % max;
    return *this;
}
QUEUE& QUEUE::operator>>(int& e)
{
    if (head == tail)
    {
        throw("QUEUE is empty!");
        return *this;
    }
    e = elems[head];
    head = (head + 1) % max;
    return *this;
}
QUEUE& QUEUE::operator=(const QUEUE& q)
{
    if (this == &q)
    {
        return *this;
    }
    if (elems != NULL)
    {
        delete[] elems;
    }
    *(int**)&elems = new int[q.max];
    for (int i = 0; i < q.max; i++)
    {
        elems[i] = q.elems[i];
    }
    *(int*)&max = q.max;
    head = q.head;
    tail = q.tail;
    return *this;
}
QUEUE& QUEUE::operator=(QUEUE&& q) noexcept
{
    if (this == &q)
    {
        return *this;
    }
    if(elems!=NULL)//判断是否为空，非空再delete
        delete[] elems;
    *(int**)&elems = q.elems;
    *(int*)&max = q.max;
    head = q.head;
```

```
    tail = q.tail;
    *(int*)&q.max = 0;
    *(int**)&q.elems = NULL;
    q.head = 0;
    q.tail = 0;
    return *this;
}
char* QUEUE::print(char* s) const noexcept
{
    int i = head;
    int j = 0;
    while (i != tail)
    {
        j += sprintf(s + j, ",%d", elems[i]);
        i = (i + 1) % max;
    }
    strcpy(s, s + 1);//将第一个逗号去除
    return s;
}
QUEUE::~QUEUE()
{
    if(elems!=NULL)
        delete[] elems;
    *(int**)&elems = NULL;
    *(int*)&max = 0;
    head = 0;
    tail = 0;
}
```

main.cpp:

```
#include"def.h"
extern const char* TestQUEUE(int& s);
int main()
{
    QUEUE q(100);
    int e;
    const char* s= TestQUEUE(e);
    std::cout << s << std::endl;
    std::cout << e;
}
```